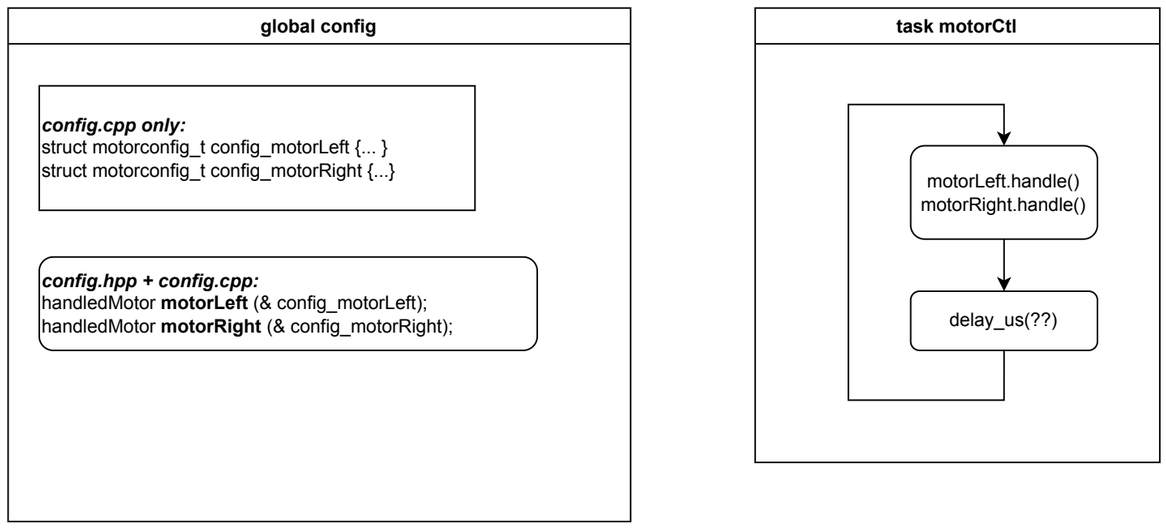
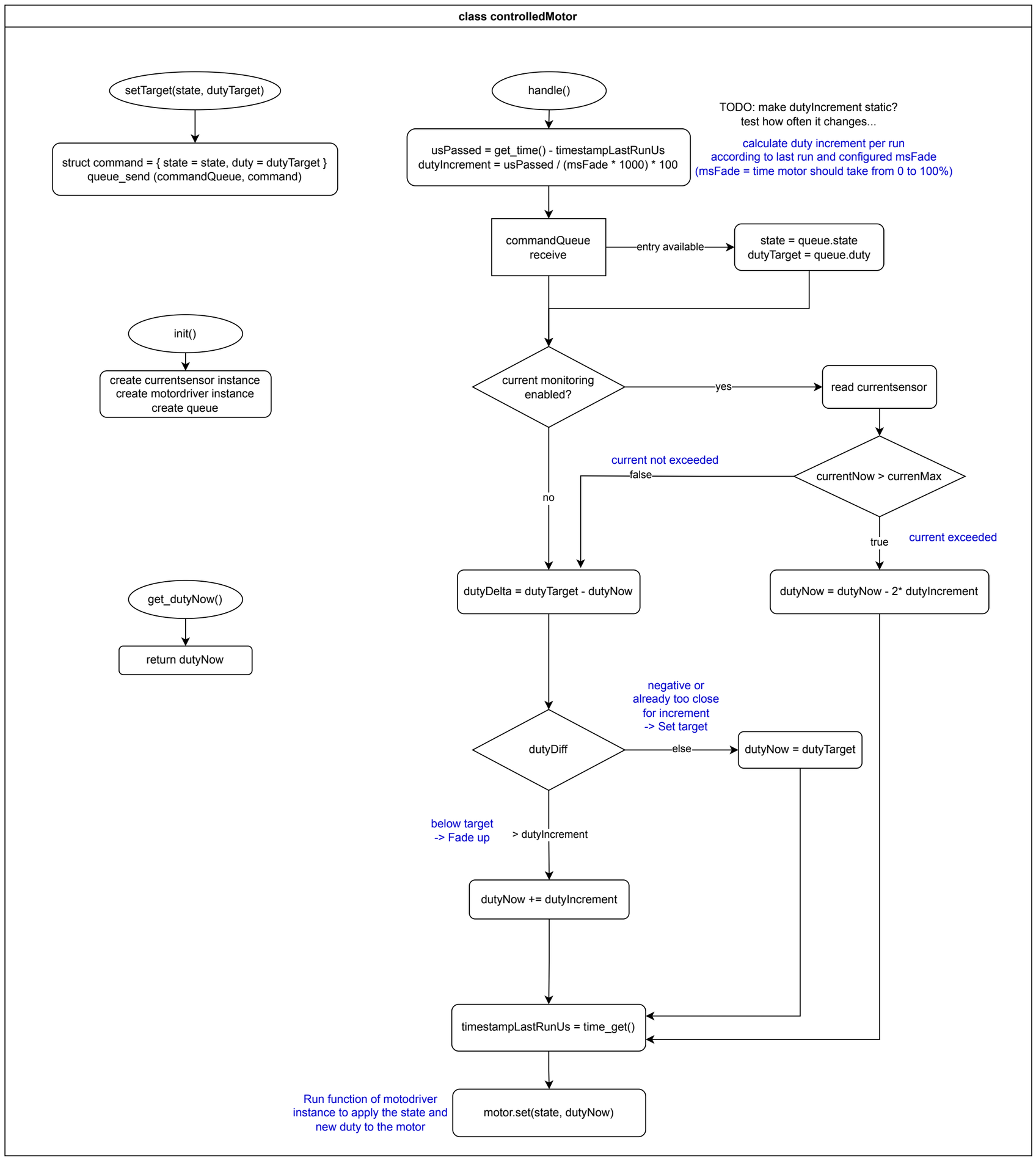
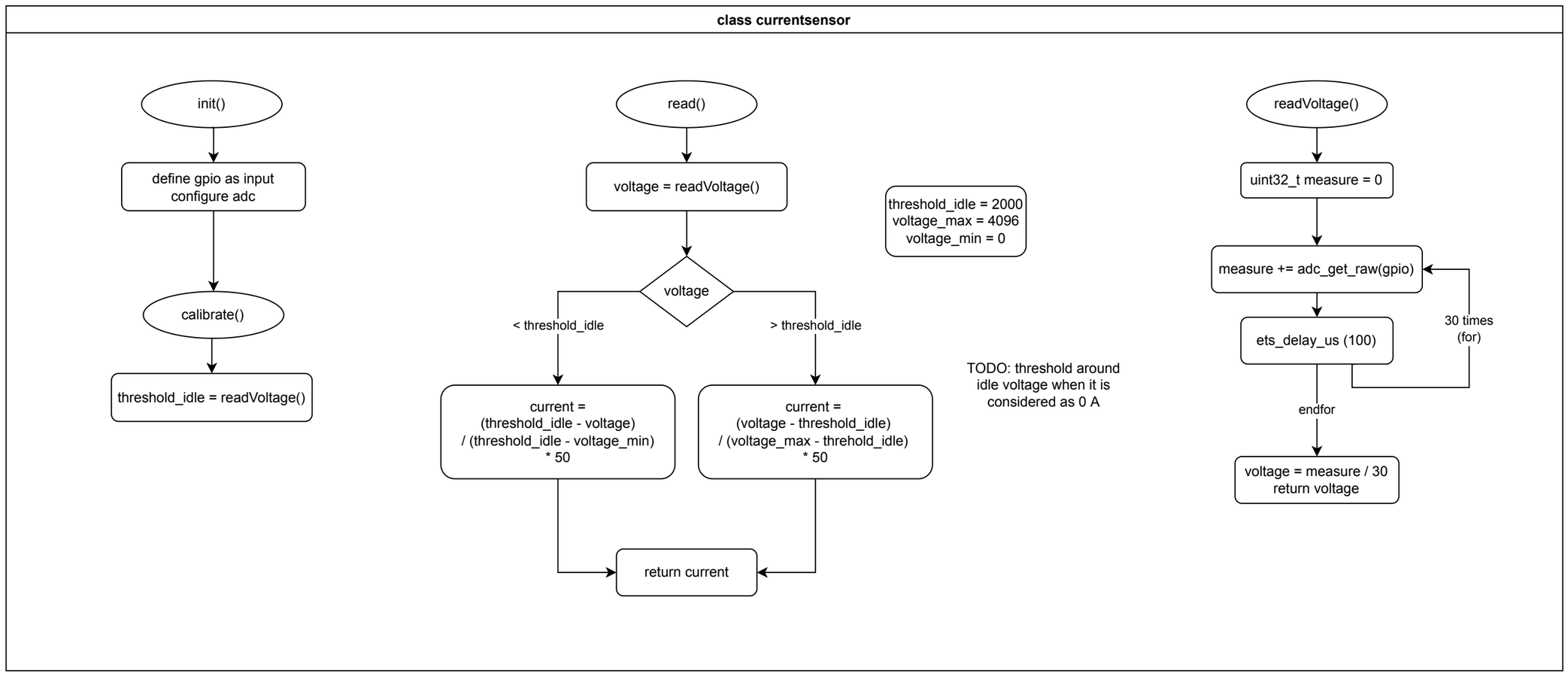

global config
config.cpp only:
struct motorconfig_t config_motorLeft {... }
struct motorconfig_t config_motorRight {...}
config.hpp + config.cpp:
handledMotor motorLeft (& config_motorLeft);
handledMotor motorRight (& config_motorRight);
task motorCtl
motorLeft.handle()
motorRight.handle()
delay_us(??)


class controlledMotor
setTarget(state, dutyTarget)
struct command = { state = state, duty = dutyTarget }
queue_send (commandQueue, command)
handle()
TODO: make dutyIncrement static?
test how often it changes...
usPassed = get_time() - timestampLastRunUs
dutyIncrement = usPassed / (msFade * 1000) * 100
calculate duty increment per run
according to last run and configured msFade
(msFade = time motor should take from 0 to 100%)
commandQueue
receive
entry available
state = queue.state
dutyTarget = queue.duty
init()
create currentsensor instance
create motordriver instance
create queue
current monitoring
enabled?
yes
read currentsensor
no
current not exceeded
false
currentNow > currenMax
true
current exceeded
get_dutyNow()
return dutyNow
dutyDelta = dutyTarget - dutyNow
dutyNow = dutyNow - 2* dutyIncrement
negative or
already too close
for increment
-> Set target
dutyDiff
else
dutyNow = dutyTarget
below target
-> Fade up
> dutyIncrement
dutyNow += dutyIncrement
timestampLastRunUs = time_get()
Run function of motodriver
instance to apply the state and
new duty to the motor
motor.set(state, dutyNow)


class currentsensor
init()
define gpio as input
configure adc
calibrate()
threshold_idle = readVoltage()
read()
voltage = readVoltage()
voltage
< threshold_idle
> threshold_idle
current =
(threshold_idle - voltage)
/ (threshold_idle - voltage_min)
* 50
current =
(voltage - threshold_idle)
/ (voltage_max - threhold_idle)
* 50
return current
threshold_idle = 2000
voltage_max = 4096
voltage_min = 0
TODO: threshold around
idle voltage when it is
considered as 0 A
readVoltage()
uint32_t measure = 0
measure += adc_get_raw(gpio)
ets_delay_us (100)
30 times
(for)
endfor
voltage = measure / 30
return voltage